# Introduction

## Objective

To generate medical reports with three heterogenous sections i.e., Findings, Impressions and Indications from IU Chest X-Ray images, highlighting the context of the particular disease i.e., location, severity and affected organs, using a combination of CNN, hierarchical LSTMs and co-attention mechanism. The project mainly consists of the following tasks:

1. Preparing an NLP Pipeline for the original findings, impressions and indication
2. Removing Contractions, Punctuations and Numbers
3. Tokenization
4. Representing the reports in word embeddings
5. Obtain the image features using a Convolutional Neural Network and the Transfer Learning framework i.e., ChexNet **[2]** (acts as the encoder here).
6. Generate the text using the labels/tags obtained from the encoder using Hierarchical LSTMs which act as the decoder in our Sequence-to-Sequence Model and substituting Attention mechanisms **[3]** to improve the encoder-decoder approach and compare the results.

## Motivation

The use of deep learning in image captioning has been a popular use case over the past few years. It is a challenging problem as it requires the machine to generate textual description from the contents of an image, similar to how a human brain would describe an image. But consider the same scenario for medical images, with the example of Chest X-ray images. For a normal human eye, chest X-ray images are just images with the skeletal and muscular features of the lungs defined in black and white. But highly trained radiologists who have studied and diagnosed various respiratory and cardiovascular abnormalities, can mark multiple areas of the images and can write down clear reports for potential abnormalities. However, to read a chest X-ray image properly, it is important that the radiologist has a thorough knowledge of the human thorax, and how various respiratory diseases might affect them. This comes with multiple years of experience by analyzing chest x-ray images with a fixed pattern of evaluation, and evolves over time depending on the history of cases a particular radiologist may handle. But, even for highly trained and experienced radiologists, writing reports is highly time-consuming especially in regions with higher population density. Looking at the other side of the spectrum, radiologists or pathologists in rural areas, with inferior imaging equipment face a similar issue. They either cannot get objective evidence to diagnose anything properly or lack the knowledge of the respiratory or cardiovascular abnormalities. Misdiagnosis of symptoms and medical errors are the third-leading cause of deaths across the world, leading to more than 3 million deaths. So, this project focuses on generating detailed medical reports using Chest X-ray images, which can facilitate the diagnosis of various respiratory and cardiovascular diseases. The dataset to be used in this project is the Indiana University Chest X-Ray (CXR) Image dataset **[1]** . It is a high resolution CXR dataset with multiple views i.e., frontal, side and posterior views. There are 7,470 images accompanying 3,955 well written reports encoded in XML. These XML reports have references to the CXR images, the findings, impressions and the indication from the CXR images. These CXR images are obtained from patients diagnosed with tuberculosis, pneumonia and various heart ailments.

## Background

Computer Aided Diagnosis (CAD) and medical imaging systems have evolved in the past decade to a point where they can partially mimic radiologists and doctors. These systems can learn and differentiate the features and abnormalities in medical images, and provide objective evidence with a higher diagnostic confidence and faster inference. In this project, we focus on generating detailed medical reports on chest X-ray images, which can be adapted later to work with other diagnostic tools such as ultrasounds and mammograms. The Indiana University dataset provides us with CXR images corresponding to various lung and heart ailments, along with well-defined reports and findings. The generation of medical reports mainly consists of two broad tasks. The first task is to treat the problem as a multi-label classification task to obtain the accurate tags for a particular image from the visual features. This is performed using Convolutional Neural Networks and a transfer learning framework called ChexNet, which is specialized for chest X-ray images. The second task is to generate the reports using these aforementioned tags, which requires the use of recurrent neural networks such as hierarchical LSTMs. To improve the quality of the sentences produced, a co-attention mechanism is also required, which makes use of the spatial information from the convolutional layers and the generated words in order to find localized regions from which the abnormalities are found.

# Literature Survey

## Survey of Existing Models/Work

In **[4]**, the authors discuss the viability of a cascade model for medical image captioning where they cascade CNNs and RNNs over multiple steps. The first step is to train the CNN with the images and predicting single object labels, and the RNN to describe their context from the text. The second step carries over the weights from the previous step and introduces a mean pooling layer in order to derive the image/text context labels from the image/text contexts of the previous step. The type of RNNs used in this framework are often used sequence generation types i.e., LSTMs and GRUs, which use the input image’s context vectors (in the form of CNN embeddings) in order to learn the annotation sequence.

The concept of using semantic attention is discussed in **[5]** where the authors first discuss the top-down paradigm (start from the high level features of the image and come up with words) and bottom-up paradigm which starts with words which describe various features of the image and combines them to form a coherent sentence using language models. Both paradigms suffer from their own weaknesses such as lack of attention to fine details in the top-down approach and the lack of end-to-end procedures from the individual features to sentences in the bottom-up approach. They suggest an idea that visual attention plays a major role of offering feedback which can help combine both the top-down and bottom-up information. Visual attention can be defined briefly as the mechanism in our visual cortex which tends to look at the low-level and semantic details of the image which it considers more important rather than the whole image. Their approach involves detection of semantic attributes using the bottom-up approach, which they call candidates, and then they employ a top-down approach in order to select which candidates should require more attention in order to yield better results. Their framework outperforms competing methods across various evaluation metrics such as BLEU and Meteor.

A similar concept of caption generation is discussed in **[6][7]** where the authors discuss the concept of scene understanding with the help of visual attention. They propose two techniques under a common framework, one being a soft and deterministic attention mechanism and another being a hard stochastic attention mechanism, which can be trained by maximizing a convergence function. A CNN extracts a 14x14 feature map, which is then processed by a RNN with visual attention over the image which provides a context vector. This vector is processed by a word LSTM which generates a word-by-word caption by utilizing a greedy search mechanism.

The approach defined by the authors in **[8]** introduces a new technique known as fine-grained label learning, to improve the quality of the generated reports in order to approach the clinically acceptable limit. They want to ensure that the broad-spectrum of the CXR findings are accurate, along with the correct anatomical location and severity. This allows the deep-learning model to pay more attention to both the coarse and fine details in the findings of a particular CXR. They assign a label to each image which comprises of Ti as the base finding type, the positive or negative finding as Ni, the core finding type as Ci and the modifiers as Mi. The team uses a specially designed parse tree in order to generate this specific labelling lexicon with over 11,000 unique terms from the reports in the training data. They use a MIMIC dataset along with the Indiana University dataset, which results in a total of 220,000 reports along with the associated images. Their technique of approaching this as a multi-label classification problem is highly performant and accurate. The images along with the associated labels are then processed by a deep learning network comprising of feature pyramids. The generated output is a pattern vector which denotes the joint occurrence of the labels in the findings, and then compared against a database of these patterns of labels which can then lead to their associated reports. This particular report is then processed in order to remove the unnecessary text or findings which is not present in the predicted label pattern.

The authors of **[9]** follow a similar approach of processing a specific CXR image using a multi-label classification technique with 14 radiographic observations including various respiratory and cardiovascular issues. They use the same dataset (Indiana University), but use a different pre-trained model to get their labels, i.e., CheXpert which trained over a large scale radiograph dataset with over 224,316 images manually annotated by 3 radiologists. This is better than using regular transfer learning techniques such as ImageNet which are trained for general purpose object recognition. They propose an encoder-decoder approach using a hierarchical LSTM (word and sentence level), as these RNN based decoders can keep the word distributions in memory, and thereby can identify patterns occurring in the training data. The encoder is built from a Resnet-152 architecture, which allows us to extract the visual features from the CXR images. They consider one input pair as one frontal image and one lateral image of one patient. The output from the encoder after average-pooling, provides us the global features from the top layers, and the local features from the final layers of the CNN. However, they highlight a significant limitation which occurs when multiple patterns of text can have comparable distributions. To combat this issue, they suggest a visual attention layer combined with a fine-tuned encoder which can recognize radiographic concepts to improve the word-level decoder.

Y. Xue et al. **[10]** have a different approach on the sentence generation by incorporating a paragraph-level RNN in order to generate topics and then providing these topics to a sentence-level RNN which can generate sentences which correspond to the final findings of the report. The image encoder is a regular CNN built on a pre-trained Resnet-152 model. Each image is divided into 196 sub-regions in order to find the global and the local features from the regions and concatenate them into a single feature vector. They also identify that the first sentence of the findings is usually a high-level description of the CXR image. So, the topic generation model as previously discussed is employed to take the global visual features learned by the image encoder and predict the topic distribution required to generate the whole sentence. The authors also train this alongside a recurrent sentence generation model, which uses the topics generated as the initial values for the recurrent model. The hidden states and the states of the LSTM cells are set as zero. A dense layer is used to transform the visual feature vector so that the dimensions are equal to the dimensions of the word embeddings. Then the visual feature vector is taken as the initial input for this network which allows us to predict the first word, the repetition of which generates the whole sentence. Finally for the sentence encoder, the authors test out the viability of a Bi-directional LSTM as well as a 2-layer LSTM. Bi-LSTM enables better encoding of the context information due to the fact that each word will correspond to two hidden states, but the 2-layer LSTM yields better results as the local visual features are taken into account along with the generated sentence. A stacked 2-layer LSTM is used as the sentence decoder, and the process is repeated till an empty sentence is obtained indicating the end of paragraph.

G. Liu et al. [11] propose a novel language generation mechanism to automate the generation of CXR image reports, with the help of reinforcement learning based on a clinical coherence. They term this as Clinically Coherent Reward, where the images will be encoded into special embedding maps, from which a sentence decoder generates topics recurrently to be used to sentence generation. Another word decoder generates a sequence from the previously generated topic along with an attention mechanism on the image. The final step is the reinforcement learning mechanism using clinical coherence from the CheXpert database, which is a rule-based annotation mechanism for 12 different diseases. The output is given as a distribution over 4 states (positive, negative, uncertain, absent) relative to the ground truth image. The complete reinforcement learning mechanism is packaged into a single loss function at the end of the decoder layer, which returns the probability distributions across the provided word embeddings, thereby improving the coherence of words in the report findings.

Background information of a particular patient is also important while preparing a medical report. X. Huang et al. [12] propose a multi-attention encoder-decoder model, which incorporates the patient’s background information using a word-embedding model and Bi-LSTM, in order to enhance the accuracy of the report. Their model follows a simple encoder-decoder architecture where the image is first encoded using a ResNet block combined with a multi-attention module i.e., it focuses on both the channel information (colors) as well as the spatial information (where the pixels are). This feature vector of the image is then used to generate descriptive reports using a hierarchical LSTM. The sentence LSTM generates a bunch of high-level vectors containing the topics which gives a semantic representation of the final sentence to be generated. The word LSTM utilizes the topic vector and the background information word embedding (discussed earlier) in order to generate a series of words thereby generating the final sentence.

An important concept that is often overlooked in a caption generation system is the correct placement of the image in the RNN-based sequence-to-sequence model. M. Tanti et.al **[13]** discusses the various methods of injecting the image’s visual features into the RNN. The first option is an early binding approach known as init-inject where the initial state of the RNN is set to be the image vector. The next option is also an early binding approach known as pre-inject where the image vector with higher level attributes is given as the first input to the RNN, along with the word vector from the caption. Par-Inject approach combines both the word vectors and image features together in a parallel manner, where a RNN is formed by putting two LSTMs in series.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Methods** | **BLEU-1** | **BLEU-2** | **BLEU-3** | **BLEU-4** | **METEOR** | **ROUGE** |
| Co-Attention **[7]** | 0.300 | 0.218 | 0.165 | 0.113 | 0.149 | 0.279 |
| FFL+CFL **[8]** | 0.560 | 0.510 | 0.500 | 0.490 | 0.550 | 0.580 |
| MVH + Attn + MC **[9]** | 0.529 | 0.372 | 0.315 | 0.255 | 0.343 | 0.453 |
| Recurrent Attention **[10]** | 0.464 | 0.358 | 0.270 | 0.195 | 0.274 | 0.366 |
| NLG + CCR **[11]** | 0.313 | 0.206 | 0.146 | 0.103 | 0.251 | 0.306 |
| Multi-Attention + BG **[12]** | 0.476 | 0.340 | 0.238 | 0.169 | 0.498 | 0.347 |

**Table 1: Comparison of various metrics across existing methods**

## Gaps identified in the Survey

Most papers in the domain use word embeddings such as GloVe word embeddings [14] which has a certain disadvantage due to the lack of medical terms in the vocabulary. The GloVe word embeddings are trained on a generic text corpus, which is not. For resolving this, BioWordVec [15] can be used which are a type of new biomedical word embeddings.

The encoder-decoder or sequence-to-sequence model, due to the lack of localized visual context, couldn’t provide accurate predictions. Most of the predictions generated by the simple Encoder-Decoder model are correct when the case is related to cardiopulmonary abnormality, but other tags are predicted wrong. This can be solved by using a Global Attention Layer **[3].**

# Project Description and Goals

# Technical Specification

## 4. 1 Functional Requirements

## 4.1.1 Product Perspective

A medical report can usually consist of various parts, which are usually heterogenous in nature. There may be images, abbreviations and complicated terminology in these reports. To avoid this issue, we will focus on three sections from a medical report i.e., findings – a large paragraph which contains keywords and parameters, indication – the doctor’s advice for the patient and impression – a sentence finalizing the results for a report. To achieve this, this application uses a multi-task architecture which works by predicting the keywords/tags (findings) by treating it as a multi-label image classification task and generate longer descriptions by using a text generating mechanism such as hierarchical LSTMs.

## 4.1.2 Product Features

The application provides us a comprehensive medical report with three heterogenous sections i.e., Findings, Impressions and Indications from Chest X-Ray images, highlighting the context of the particular disease i.e., location, severity and affected organs, using a combination of CNN, hierarchical LSTMs and co-attention mechanism.

## 4.1.3 User Characteristics

The intended audience for this application consists mostly of medical professionals and radiologists. The users of the application should be aware of the all the different functions of the application and the proper procedure to use the application.

## 4.1.4 Assumptions & Dependencies

It is assumed that the dataset and the associated reports being used to implement the application is publicly available with all the required legal permissions. The images and reports have been thoroughly checked in order to redact any personal patient data in order to maintain their privacy. The application’s accuracy partially depends on the provided data (both training and testing) as well as the data provided by the intended users.

## 4.1.5 Domain Requirements

Due to the critical nature of medical expert systems, the domain requirements mostly consist of adherence to patient-privacy laws, and patient-disclosure laws, along with regulations to check the diagnosis and the accuracy of the reports generated. Although the objective of the project is to reduce the human involvement in generating the medical reports, during the initial release of the system, it would require heavy involvement from radiologists and medical professionals in order to reduce the amount of misdiagnosis or critical mistakes.

## 4.1.6 User Requirements

The intended audience of the application would have some essential requirements, such as proper user management, high visibility and contrast on the user interface suitable for all environments and complete transparency of the data and the predictions while maintaining the privacy of the patients involved. The application would require a well-prepared documentation and tutorials for the intended users.

## Non-Functional Requirements

## 4.2.1 Product Requirements

**4.2.1.1 Efficiency**

The application should be computationally efficient and the cost of working on larger datasets should be proportional. The efficiency can be greatly improved by parallelizing the training and testing of the deep learning models as they are the ones which require the most amount of computational time and resources.

**4.2.1.2 Reliability**

Due to the sensitive and critical nature of medical expert systems (especially involving diagnostics), it is necessary that the application is highly reliable and accurate. The complete application can be deployed as a web application on highly available servers (like AWS or GCP) to be accessed from anywhere. An alternative could also include packaging this as a standalone application for offline support which might be beneficial in rural areas where connectivity might be an issue.

**4.2.1.3 Portability**

Machine Learning models can be easily deployed online using micro-web frameworks such as Flask and FastAPI which are intended to be used on Python. TensorFlow along with Keras allows us to save the model weights in order to quickly model predictions on the given data. TensorFlow also allows us to convert these models in order to achieve compatibility with Android/iOS mobile applications and JavaScript. So, it’s entirely possible to deploy this application as a web application making it extremely portable, with the ability to run on multiple platforms.

**4.2.1.4 Usability**

With a well-prepared documentation, FAQs and tutorials, the application will be usable and convenient for every intended user. Future revisions can include multilingual support in order to improve usability among a wide variety of users. However, the multilingual support for the medical reports would require a highly accurate language translator or an accurate dataset with the reports in the language to be supported. This is due to the fact that medical reports are highly critical and sensitive in nature.

## 4.2.2 Operational Requirements

* **Economic –** Theapplication should be economically viable in terms of modelling predictions and should be cheap and efficient when the application needs to be scaled.
* **Social –** The application should be socially responsible in terms of the data and the predictions generated from it. The application and the necessary data should be publicly accessible.
* **Political –** The application should be unbiased and free of any political propaganda.
* **Ethical –** The application should respect the personal privacy of the users and provide unbiased results irrespective of race or gender.
* **Health and Safety –** Due to the application being an important part in the medical system, it is important that health and safety regulations are strictly followed, which may require validation from regulatory authorities. The regulatory checks will be to check the viability and the accuracy of the predicted results due to the sensitive and critical nature of the task at hand.
* **Sustainability –** The application should be easy to update to the latest versions in order to provide the intended features without any interruption to the users of the applications.
* **Legality –** The datasets used to train the model should be open-source or publicly available along with the guarantee that the provided information in the images and reports are correct and validated by proper authorities, given that medical reports are highly critical. Any personal data in the datasets as well as the application should be redacted or encrypted as per requirements.
* **Inspectability –** Proper measures should be taken in order to facilitate any debugging in case of any errors, along with a flexible and modular design architecture which can be inspected and validated by independent as well as government authorities.

## System Requirements

## 4.3.1 H/W Requirements

The model was trained over a Colab Pro instance, which has the following specifications. The processor is a server-grade Intel Xeon processor with support for multithreading which helps with the initial preprocessing and streaming of the data during the training phase. The GPU is one of the most essential components when training deep learning models as it allows us to distribute the massive amount of mathematical computation amongst many cores. The Nvidia **Tesla P100** is the world’s most advanced GPU for HPC and Deep Learning workloads. It has a **16GB of HBM2** memory along with **3584 CUDA cores**. The Colab instance also has **28 GB** of usable RAM and **disk storage of 150GB** which provides us plenty of memory in order to retain the model data, weights and various other information.

However, for using the deployed web application or to use the desktop application, the hardware requirements can be reduced i.e.

|  |  |
| --- | --- |
| **Processor** | Intel Core i3 family / AMD Ryzen 3 |
| **RAM** | 4 GB DDR3 |
| **GPU** | Integrated Intel Graphics / AMD Radeon |

**Table 2: Hardware Requirements**

## 4.3.2 S/W Requirements

The software requirements including the programming languages, libraries/frameworks and dependencies are listed below. The features and the reasons to choose these are also discussed later.

|  |  |
| --- | --- |
| **Operating System** | Windows, Linux, MacOS |
| **Programming Language** | Python |
| **Data Libraries** | NumPy, Pandas, Dask, Scikit-Learn |
| **Web Framework (Deployment)** | FastAPI |
| **Neural Network Libraries/Frameworks** | TensorFlow, Keras |

**Table 3: Software Requirements**

**4.3.2.1 Python**

Python is a high-level programming language that attracts most data scientists and machine learning engineers due to its **enhanced readability** and **concise code** even for the most complex algorithms. This simplicity allows developers and engineers to solve the particular problem at hand rather than focusing on the complicated syntax of a particular language. Developed by Guido Van Rossum, the first version of Python was released in 1991 in order to replace the C programming language and its derivatives, due to its flexibility offered by Python in terms of programming paradigms. Python is widely known as an object-oriented programming language with **dynamic types** and **built-in garbage collection**. But it also supports structured programming techniques such as procedural programming and functional programming. It’s **easy and intuitive to learn** due to which it is consistently ranked as one of the most popular programming languages in the world.The comprehensive standard libraries included with Python serves most of the basic needs of most programmers. But due to the popularity of Python among developers around the world, it has various **open-source libraries and frameworks** which can be easily utilized by the package management tool that accompanies most Python installations – **pip.** Pip allows developers to install packages from the Python package index which acts as an online repository for public Python packages. Finally, the reason why Python is so popular with data scientists is due to its portable and extensible nature. It can run on multiple-platforms (Linux, Windows, MacOS) and can be easily installed and deployed on virtual machines.

**4.3.2.2 Anaconda**

Anaconda is a **free and open-source packaged distribution** / **package manager** for Python and R which is especially useful in the field of scientific computing (data science, big-data processing, machine learning applications, predictive analytics etc.). The base distribution (conda) comes with over 1,500 essential packages bundled into one installer along with a GUI (Anaconda Navigator) and a dedicated CLI (Command Line Interface) which helps in installing new packages, maintaining and updating existing packages. Anaconda was initially developed to **simplify the installation and management of dependencies** which was a huge obstacle for most data scientists and engineers who were working with Python and pip. Anaconda has a unique way of dealing with package versions in order to **avoid conflicts** with previously installed packages. Anaconda analyzes the current development environment to identify all the installed packages, and takes in the **version limitations** specified by the user and it works out the installation of the package from the **Anaconda repository**. Also included is a GUI known as the Anaconda Navigator which allows the users to manage the applications installed alongside Anaconda. These include essential IDEs and interactive data visualization tools such as JupyterLab, Jupyter Notebook, Spyder and Visual Studio Code.

**4.3.2.3 Jupyter Notebook**

Jupyter Notebook (previously known as IPython Notebooks) is an **interactive web-based environment** which is commonly used for scientific computation. It is a **REPL i.e., Read Eval Print Loop** that can be opened in a browser which allows the user to execute a small section of code. These small sections of code are known as cells. A Jupyter Notebook usually contains **multiple formats of data** combined together (code, text, mathematical plots and media elements) into a single JSON document, and uses an extension. ipynb. It **enhances the usability and readability** of code and developers can divide the code into sections and these notebooks can be converted into other document formats including HTML, LaTeX, PDF etc. A Jupyter Notebook works by connecting to a kernel which could be running any one of the allowed programming languages (in our case Python), which allows the instantaneous execution of code present in Jupyter Notebook cell. These kernels can either be in the same machine or can be configured to **use remote servers** with higher computation capacity in order to reduce the time required to execute the code. Online services like Colab (by Google) provide free or subscription-based services in order to run Jupyter Notebooks on shared virtual machines especially designed with high computational capacity (more RAM, better GPUs and CPUs) in order to reduce training times.

**4.3.2.4 NumPy and Pandas**

NumPy is a popular scientific library for Python which adds support for large data structures which are highly essential for **high-level mathematical calculations**. The support for optimized multi-dimensional arrays improved the performance of Python in mathematical computation, and the use of linear algebra libraries such as LINPACK enabled higher efficiency in **linear algebra computations**. The ndarray data structure is widely used in libraries such as OpenCV which stores images in the form of NumPy arrays, thereby allowing **more efficiency in image processing** tasks such as slicing and masking as it is easier to access the specific pixels in an image.

Pandas is also a popular library for python which offers support for large data structures known as **DataFrames** and the operations to manipulate these data structures which may be numerical tables or timeseries objects. These dataframes can be read from various file formats including CSV, JSON, SQL, Pickle, Excel files etc. and can be written to these formats as well. It is **highly optimized for performance** as it was primarily developed to support quantitative analysis on large amounts of financial data. Now it is mainly used for exploratory data analysis and preprocessing datasets due to its readily available table manipulating functions **for selecting, reshaping, merging, joining and slicing of data**. This project extensively uses Pandas as the report data is converted into a single dataframe and stored in the Pickle format. This allows us to retrieve the training data faster from the stored file and manipulate it according to our needs.

**4.3.2.5 Scikit-Learn**

Scikit-Learn or sklearn is a popular **machine learning library** which contains **prebuilt functions** for various baseline machine learning tasks such as **preprocessing** (scaling, encoding etc.), **regression**, **classification** and **clustering**. It also supports functions such as gradient boosting and it is highly compatible with previously discussed numerical libraries such as NumPy. This project uses sklearn in the preprocessing phase in order to scale the numerical features to be compatible with neural networks i.e., between 0 and 1, and encoding categorical features using numerical values (tags in the CXR reports).

**4.3.2.6 Dask**

Dask is a popular open-source library which allows you to add **parallel computing** capabilities to your data-science or machine learning application without any major rewriting of code. This is due to its **flexible task scheduling technique** which is highly optimized for computational workloads to be run on large clusters or single workstation. It is **highly scalable** and contains a data structure collection which is specifically designed to run on parallel computation. Arrays from NumPy and DataFrames from Pandas can be readily replaced with the **Dask Arrays** and DataFrames respectively which allows you to distribute the data and the tasks associated with this data to be distributed among multiple physical or virtual machines. Usually, the NumPy and Pandas tasks are limited to few threads of the CPU, but using Dask you can utilize the entirety of the CPU’s threads easily in order to process your computation faster. The project uses Dask DataFrames along with Pandas in order to process the large amount of data in a small amount of time. Dask also has support for Keras and TensorFlow in the form of wrappers which allows the training of a model to be executed in parallel using multiple machines. Dask also has a preprocessing library which allows us to **execute transformation functions** similar to Scikit-Learn. The data is distributed into parallel threads known as **Pipelines** and the fit and transform process is completed faster.

**4.3.2.7 FastAPI**

FastAPI is a **micro web-framework** developed for Python, which is widely used to **deploy machine learning models** in the form of REST APIs. A replacement for the Flask framework, FastAPI is based on ASGI (Asynchronous Server Gateway Interface) which allows the server to handle a particular request by branching into a separate thread thereby not blocking the main server thread. This increases the number of requests that can be served using the same server. It offers faster JSON serialization and deserialization due to the underlying frameworks of pydantic and starlette, which allows the server to process predictions which may receive large amounts of input data.

**4.3.2.8 TensorFlow**

TensorFlow is a popular open-source library which was developed by Google to specifically focus on the **development of deep neural networks**. TensorFlow primarily works using specialized multi-dimensional data-arrays known as **tensors**. Tensors represent multilinear relationships and form an integral part of computer science, physics, mechanics, electrodynamics etc. These tensors can be integrated with parallel processing techniques such as CUDA in order to **reduce computational time by using GPUs** to train neural networks. GPUs have a greater number of cores, so TensorFlow allows us to take advantage of GPU based systems in order to train, test and deploy deep learning projects faster. It allows us to easily build models due to the **multiple abstraction levels** provided by the library along with Keras (discussed later). The Model API is feature-packed to build **state-of-the art models** and create complex topologies using features such as eager execution which allows us to prototype faster. TensorFlow also provides robust APIs (TensorFlow Serving) which allows us to **save trained models** into files with the model weights. These model weights can be used to reconstruct the model on a web server, which allows us to **deploy these models** over REST APIs in order to serve the predictions on high-performing production environments. It also provides mini-libraries for various hardware levels in order to deploy the trained models. Tensorflow.js is a simple JavaScript Library which allows us to create basic machine learning applications or use pre-trained deep-learning models in the comfort of the browser.

**4.3.2.9 Keras**

Keras is another open-source Python library which provides support for easily developing and **training deep learning models**. Previously a separate library, it became a part of TensorFlow v2, where it can be accessed through the **tf.keras** module. Built on top of the TensorFlow’s low-level tensors, it improves the efficiency and the scalability of the models by introducing pre-built and **optimized layers** with completely **customizable hyperparameters**. It can be visualized as an infrastructure layer, which provides core functions such as sequential modelling using APIs such as **layers** and **models.** It provides direct techniques to fit the training data and predict outputs using the model.fit() and model.predict() functions respectively. Keras also provides **support for Recurrent Neural Networks** providing us with RNN layers such as LSTM and GRU, which are highly essential for this project. They form the base in our encoder-decoder architecture. Keras also provides **support for transfer learning** which helps us in importing a pre-trained model trained on more amount of available data, and thereby lowering the computation time required and increasing the accuracy of our program. In this project, a pretrained ChexNet model is used to identify essential features from the CXR image which allows our encoder-decoder models and attention models to identify the regions of interest and predict the necessary tags for the text generation.

# Design Approach and Details

## Introduction and Related Concepts

A medical report can usually consist of various parts, which are usually heterogenous in nature. There may be images, abbreviations and complicated terminology in these reports. To avoid this issue, we will focus on three sections from a medical report i.e., findings – a large paragraph which contains keywords and parameters, indication – the doctor’s advice for the patient and impression – a sentence finalizing the results for a report. To achieve this, we propose a multi-task architecture which works by predicting the keywords/tags (findings) by treating it as a multi-label image classification task and generate longer descriptions by using a text generating mechanism such as hierarchical LSTMs. Hierarchical LSTMs [16] are specialized recurrent neural networks which are often used for text generation from images and video frames. It is built to consider both high-level language features from the training text and low-level visual features obtained from the processed image.

The dataset to be used in this project is the Indiana University Chest X-Ray (CXR) Image dataset. It is a high resolution CXR dataset with multiple views i.e., frontal, side and posterior views. There are 7,470 images accompanying 3,955 well written reports encoded in XML. These XML reports have references to the CXR images, the findings, impressions and the indication from the CXR images. These CXR images are obtained from patients diagnosed with tuberculosis, pneumonia and various heart ailments.

## Architecture for the Proposed System

The architecture of the proposed medical report generation system can be divided into three distinct parts:

**5.2.1 NLP Pipeline**

The findings, impressions and the indications obtained from the reports have to be properly cleaned to be used in the model. This involves the following steps:

1. Converting all characters into lowercase
2. Removing contractions from the text e.g., won’t – will not, can’t – cannot.
3. Removing punctuation from text with the exception of full stop, as the findings from the reports may contain more than one sentences.
4. Removing all numbers and redacted data from the text.
5. Removing smaller words and adverbs with the exception of “no” as it adds significant value. e.g., adverbs such as “there”, “then”.
6. Tokenization and addition of identifier tokens such as “\_start” and “\_end” tokens which are necessary in the text generation process.

**Examples –**

Findings before the cleaning process:

No pleural effusion or pneumothorax. No acute bone abnormality.

Findings after the cleaning process:

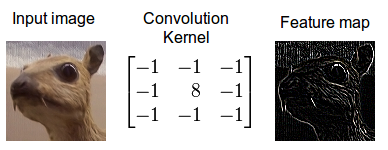
***\_start*** no pleural effusion no pneumothorax. no acute bone abnormality. \_***end***

**5.2.2 Convolutional Neural Networks**

A convolutional neural network (CNN) is a special class of neural networks which are primarily used to work with visual data. Introduced in late 1980s by Dr. Yann LeCun, it was inspired by the work of a Japanese scientist Kunihiko Fukushima who had invented a basic neural network to recognize images, known as neocognitron. The first CNN was used to identify handwritten digits (LeNet) and was deployed to read postal codes and bank cheques in the early 1990s. The working of a CNN is very similar to the human visual cortex.

A CNN usually has multiple layers. A typical convolutional layer will multiply the image (pixel-wise) with multiple predefined filters in order to identify the features (lines, edges, shapes) in particular image. This operation is known as convolution and often requires a padded image. The padding options include zeros, ones and replication of the pixels, or mirroring of pixels. The initial convolutional layers extract low-level information from the image i.e., edges, gradients and colors. The subsequent layers enable the CNN to identify the high-level features of the image as well. The convolutional layers are often combined with Pooling layers. The main objective of a pooling layer is to extract the dominant features from the image and reducing the computational resources required to process the data further. The most popular pooling layers include Max Pooling (provides the maximum value of the pixels from the assigned section) and Average Pooling (provides the mean value of the pixels from the assigned section).

The final output from the final pooling layer is then fed to a flatten layer which is used to convert the given image into a one-dimensional feature vector. This feature vector is then processed by a Dense Layer (Fully Connected Layer) and then is followed by an output layer which usually houses a softmax activation function in order to provide a probability distribution of the image relative to the associated image classes.



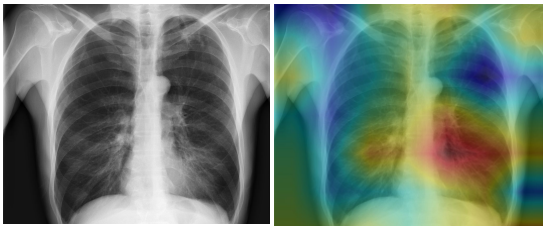
**Figure 1: Convolution Process Visualized**



**Figure 2: Max Pooling and Average Pooling**

In order to extract the visual features of the image in this project, we make use of convolutional neural networks. The CNN acts as the encoder in our model, and provides us a feature vector with the visual features of the image, which can be used to predict the tags for that particular X-ray. This is done by considering the problem as a multi-label classification task where the output layer provides you with the probability of a set number of tags. These predicted tags help us massively in text generation which is discussed in the next section. However, the size of our dataset (7,470 images) is not enough to train a CNN properly. To remove this bottleneck, we have to use a transfer learning framework. Most transfer learning frameworks such as VGG16 or InceptionV3 are trained over generic image datasets which doesn’t serve our purpose.

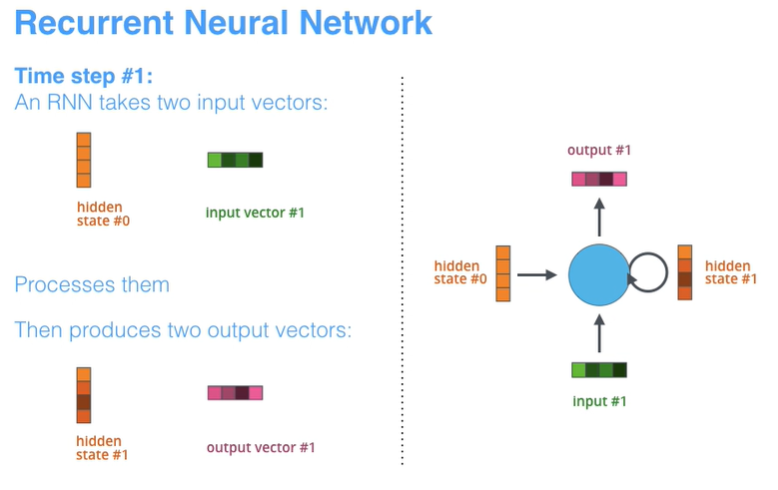
Fortunately, ChexNet is a convolutional neural network especially trained on Chest X-ray images. It was trained over 1,12,120 images and contains 121 layers where the input is a chest X-ray image, and the output is the probability of 14 different diseases along with a localized heatmap which highlights the visual features of the chest x-ray image. However, we do not need to classify the image into one of those 14 categories, so we can remove the final classification layer. From an image of dimensions (224,224,3), we get a feature vector with a length of 1,024. We have two images associated with a report, so we concatenate the two feature vectors to get a feature vector with length 2,048. This final feature vector will be passed along with the report to the decoder which is discussed in the next section.

****

**Figure 3: Chest X-ray image from the dataset before and after passing through the CNN**

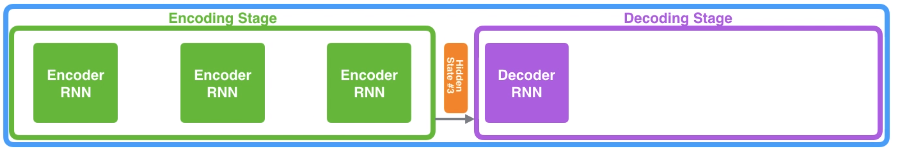
**5.2.3 Hierarchical LSTMs and Attention Mechanism**

The text generation process of the application is handled by Recurrent Neural Networks (RNN). Recurrent neural networks are a special class of neural networks which usually work with temporal/sequential data. RNNs can be considered as a directed graph with the connections representing a time sequence. The working process of a RNN is pretty simple; it takes two input vectors where one of them is known as the **hidden state** of the RNN in that time step. After a particular time-step, the hidden state of the RNN is updated and it generates an output vector. The next time-step will utilize its hidden state (which has the temporal context stored from the previous time step) and the next input vector to generate the next output vector. This is depicted in Figure x. The hidden state opens up a storage mechanism for the RNN which can be additionally replace by other networks/cells with feedback mechanisms and time delays. These controlled states form a part of Long Short Term Memory (LSTMs) or Gated Recurrent Unit (GRU), which are also known as Feedback Neural Networks.

****

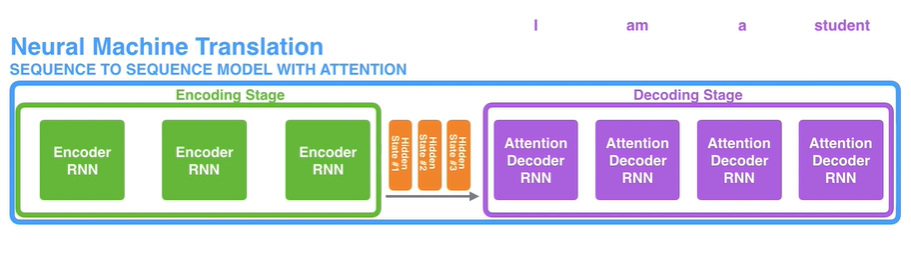
**Figure 4: RNN Timestep Visualized**

However, when there are multiple inputs involved (image and text), a Sequence-to-Sequence model is better suited for the purpose. A sequence-to-sequence model consists of an encoder and a decoder where both are based on RNNs. A simple encoder and decoder network can be built with a series of LSTM cells. The encoder network works by understanding the input sequence and returns a reduced representation of it, in the form of a feature vector. The decoder network uses this feature vector and generates an output sequence according to its own understanding. However, at each time step of the decoder, the LSTM cells generate the probability distribution of the next word as the output, therefore it needs to make a decision on which word should be next in the sequence. This could be done greedily by figuring out the word with the maximum probability at each time step. This approach, though handy could yield inaccurate results in some cases. So, a better approach would be to use a beam search which takes the probability of next k words and chooses the best approach among the choices to maximize the probability.

****

**Figure 5: Seq2Seq Model without Attention Mechanism**

The regular sequence to sequence model however suffers from a disadvantage in the image captioning problem. The regular seq2seq model passes the final hidden state from the encoder to the decoder, which means the intermediate hidden states are lost thereby losing the context information from the intermediate states. To fix this, we can store the intermediate hidden states [**17]**, to greatly improve the quality of the generated caption, as we can recover the lost contextual information. This is known as Attention mechanism. (Fig d ).It is analogous to how humans pay attention to specific parts of a text or an image to get the most important sections/regions in the text or the image at a particular time. To implement the attention mechanism, the hidden states are assigned weightages according to the importance of the contextual information. Now, that our training data contains both images and text, the attention mechanism needs to be adjusted to simultaneously understand both the text content and the visual content of the data. Z. Yu et al **[18]** define a co-attention approach which is used for visual question answering. The attention distributions for the text content and the visual content are learned separately as context vectors. The visual context is learned from the CNN (ChexNet features, the textual context is learned from the encoder architecture. The decoder has to be designed in such a way to merge both the visual and textual context in order in improve the accuracy of the results.

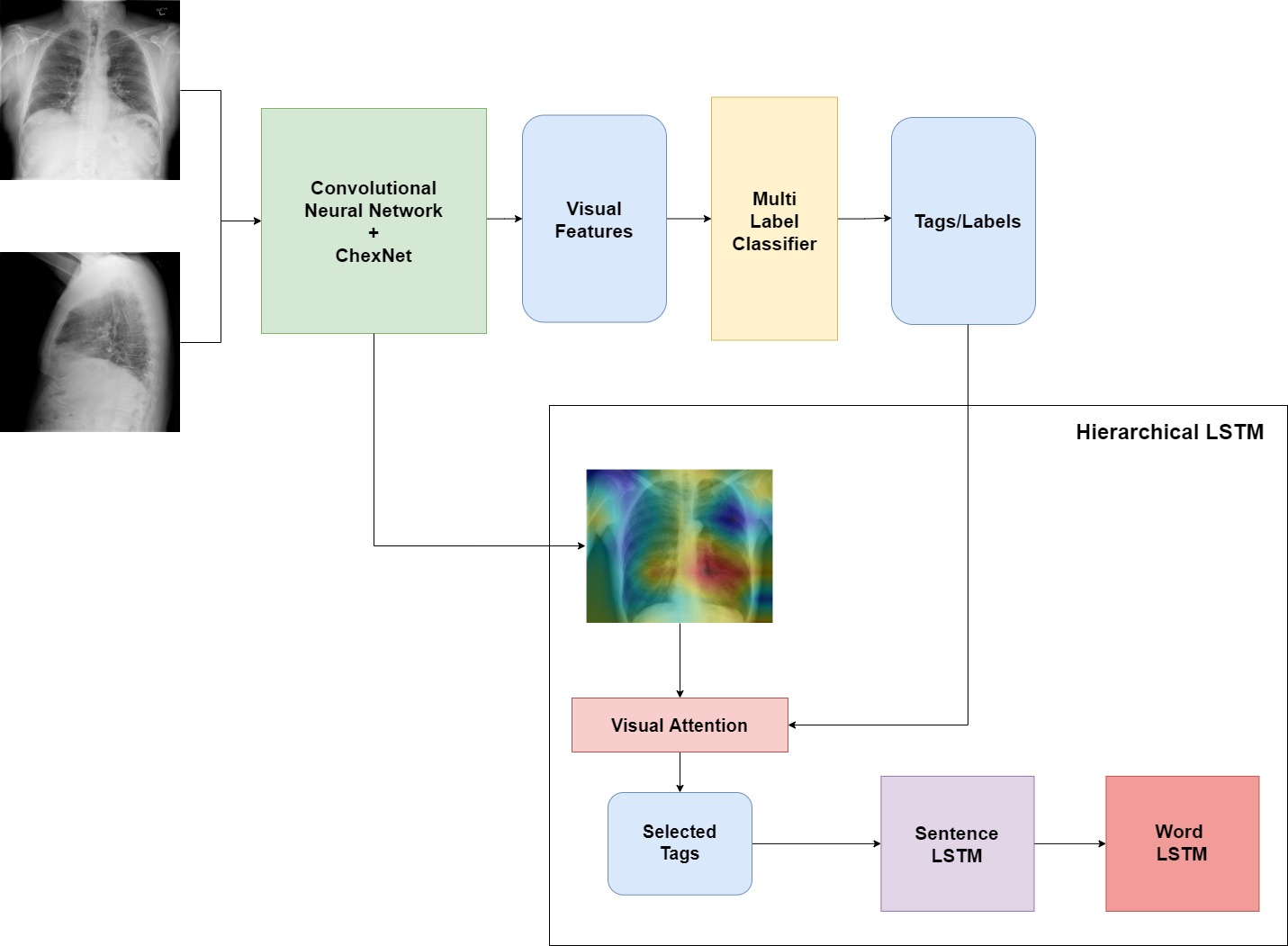
****

**Figure 6: Seq2Seq Model with Attention Mechanism**

Hierarchical LSTMs **[16]** are specialized recurrent neural networks based on the encoder-decoder architecture which are often used for text generation from images and video frames. Here, it is built to consider both high-level language features from the training text and low-level visual features obtained from the processed image. Note that the keywords/tags obtained from the image are generated by the fully connected layer, which results in a loss of spatial information. To improve these results, an additional mechanism known as Co-Attention is added. Co-Attention mechanism uses the spatial information from the visual features of the convolutional layers and the semantic features obtained from the tags of the specific image (which are generated by the fully connected layer).

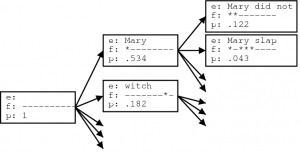
When the visual features and the tags arrive at the decoder, the high-level spatial information provided by the localized heatmap help us to focus on the tags which are visually highlighted more, and yield better results. This new context vector with the embeddings of the selected tags is passed on to a sentence LSTM. A Sentence LSTM will generate multiple sentences as suggestions to the provided words, using a technique known as beam search which predicts the probability distribution across the given vocabulary and returns the words which have the maximum probability to be subsequent words in the sentence. Beam search selects multiple alternatives for an input sequence, based on conditional probability and a parameter known as beam width. When the sentence vector is successfully produced, it is passed on as a context vector to the Word LSTM. Word LSTM employs a greedy search mechanism, which selects a single candidate which is suitable for the input sequence in a time step. This improves the quality of the final sentence generated.

## Proposed System Model

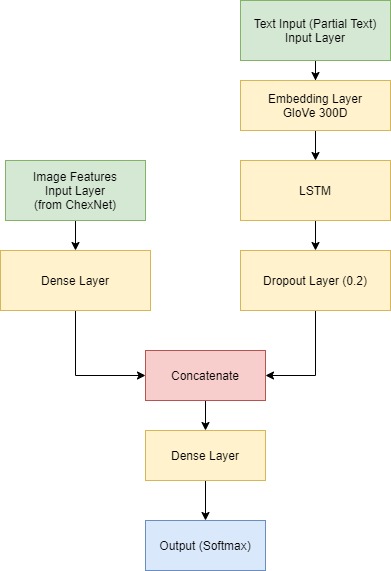


**Figure 7: Architecture of the medical report generation system**

* 1. During the training phase, the preprocessed reports from the NLP Pipeline along with the two associated images (training data) form the inputs to the neural network.
  2. The images are then augmented in order to avoid overfitting and increasing the amount of data available to the CNN. This is done using the Image Augmentation utilities which is available in the TensorFlow library. The text is converted into their embedding form using a GloVe embedding (300 dim).
  3. The images are then passed through the ChexNet CNN along with a few custom CNN layers which allows us to get a visual feature context vector. The visual features of both the images are concatenated and sent to the visual attention layer. This is the encoded context vector in our sequence-to-sequence model.
  4. The visual features are also sent to a multi-label classifier which predicts the tags/labels from the given images which are sent to the input layer as partial reports.
  5. Once the inputs are encoded, they are passed to the hierarchical LSTMs which are specialized RNNs used for image captioning. There are multiple methods for injecting an image in an RNN as mentioned in **[13]**, but the merge architecture suits our use-case the best. Using the merge architecture ensures that our RNN is not exposed to the visual features until the partial reports are prefixed. Due to this late binding technique, the RNN does not modify the image representation with every time step.
  6. The RNN Outputs are regularized using a dropout layer, and the “merged” with the normalized image input vector, which is then passed through the dense layer. The output layer has a SoftMax function which then generates the probability distribution across the words present in the vocabulary.
  7. The final report is generated using a beam search mechanism by the hierarchical LSTMs (process explained above)



**Figure 8: Beam Search to find subsequent words**



**Figure 9: Depicting Merge Architecture for report generation**

# SCHEDULE, TASKS AND MILESTONES

# 6.1 Schedule

Schedule diagram here

# 6.2 Tasks

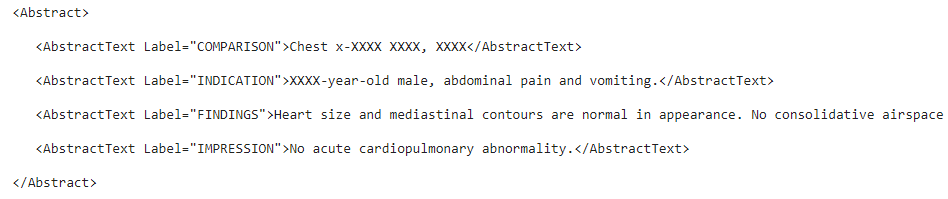
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Task Name** | **Duration (in days)** | **Dependency** | **Description** | **Status** |
| 1. | Background Research | 3 | NIL | To learn more about the existing work in the area of image captioning and chest X-ray image features. | Completed |
| 2. | Exploring the data | 3 | NIL | The Indiana University dataset contains large number of images, reports and metadata in the form of XML reports. Exploring the data will improve the understanding of the problem. | Completed |
| 3. | Extracting usable datasets | 4 | 2 | The metadata (MeSH tags) and the findings from the XML reports had to be extracted using Python scripts. | Completed |
| 4. | Literature Review | 6 | NIL | Reviewing Papers from the last 5 years regarding medical captioning, Chest X-ray CNNs and co-attention mechanisms | Completed |
| 5. | Initial Architecture Design | 2 | NIL | The initial architecture design of the complete application | Completed |
| 6. | Preprocessing Text Data | 2 | NIL | The text data (findings and tags) had to be preprocessed. The process includes text cleaning and tokenization | Completed |
| 7. | Faculty Review 1  (12/3/21) | 1 | NIL | Faculty review – basic explanation of the project, literature review and the objective of the project | Completed |
| 8. | Preprocessing Image Data | 3 | NIL | The images had to be preprocessed and augmented to reduce overfitting on training data and increase the training samples | Completed |
| 9. | Input Data Pipeline | 2 | NIL | The input data pipeline combines the augmented images (frontal and lateral X-rays) with their corresponding reports. | Completed |
| 10. | CheXNet Implementation | 2 | NIL | Transfer Learning model with multiple layers. Required Trial and error to test which output layer will provide best results for the visual feature vector. | Completed |
| 11. | Image Encoder Architecture | 4 | 10 | The image encoder architecture utilizes the CheXNet model, along with custom layers to encode the image into the visual vector | Completed |
| 12. | Text Encoder Architecture | 4 | NIL | The text encoder takes the image caption, converts into the text embedding and obtain the encoded text vector. | Completed |
| 13. | Basic Sequence-to-Sequence Model | 3 | NIL | A basic seq2seq model with merge architecture. | Completed |
| 14. | Faculty Review 2  (22/4/21) | 1 | NIL | Completed 75% of the implementation along with demo | Completed |
| 15. | Seq2Seq model with co-attention | 3 | NIL | Seq2Seq model with co-attention mechanism. Obtained better results | Completed |
| 16. | Metrics Comparison | 2 | NIL | BLEU, METEOR and ROUGE scores | Completed |
| 17. | GUI Development | 4 | NIL | GUI development and deploy on Streamlit. | Completed |
| 18. | Final changes and Documentation | 10 | NIL | The completion of the final report and the presentation | Completed |

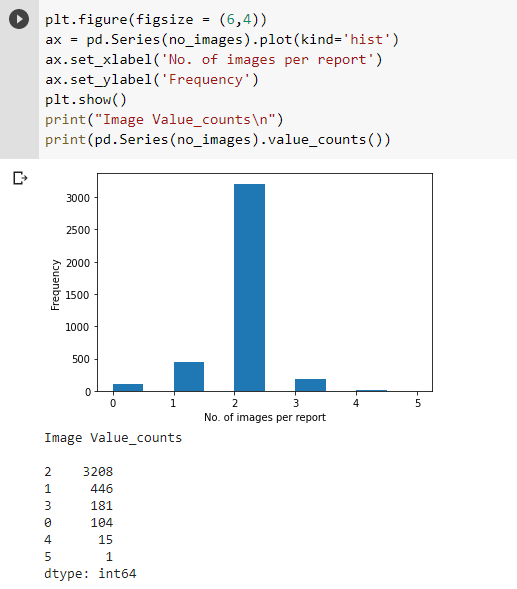
# 6.3 Milestones

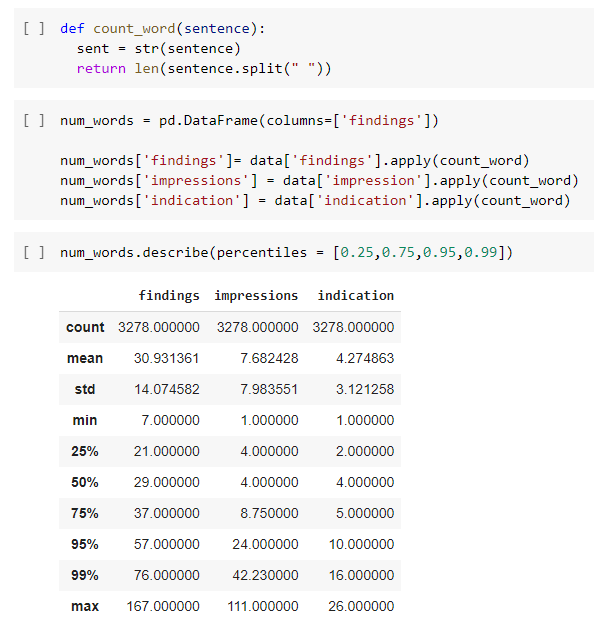
|  |  |  |
| --- | --- | --- |
| **No.** | **Milestone** | **Date** |
| 1 | Dataset Extraction | 03/02/21 |
| 2 | Architecture Design | 15/02/21 |
| 3 | CheXNet integration and tags generation | 28/02/21 |
| 4 | Image and Text Encoding | 05/03/21 |
| 5 | One Step Decoder (Seq2Seq Model) | 06/04/21 |
| 6 | Co-Attention Mechanism | 13/04/21 |
| 7 | GUI Completed | 01/06/21 |
| 8 | Documentation completed | 03/06/21 |

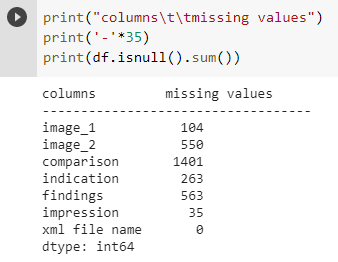
# PROJECT DEMONSTRATION

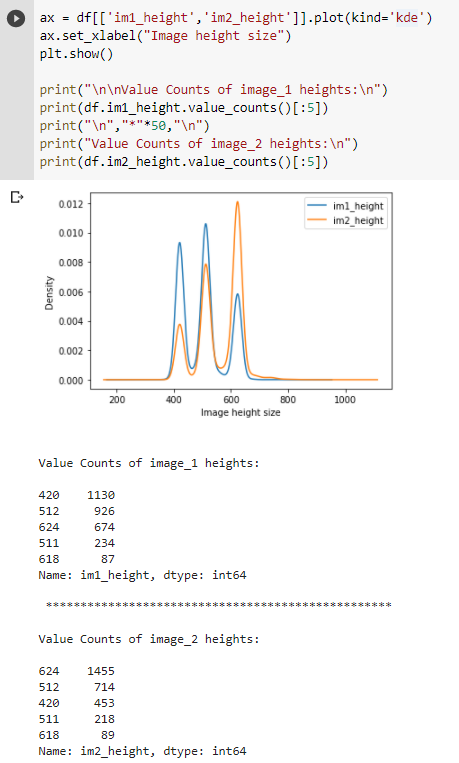
**7.1 Exploratory Data Analysis**

****

****

****





**7.2 Text Cleaning**

def decontracted(sentence):

    # removing contractions

    sentence = re.sub(r"\'s", " is", sentence)

    sentence = re.sub(r"\'d", " would", sentence)

    sentence = re.sub(r"\'ll", " will", sentence)

    sentence = re.sub(r"\'t", " not", sentence)

    sentence = re.sub(r"\'ve", " have", sentence)

    sentence = re.sub(r"\'m", " am", sentence)

    sentence = re.sub(r"won\'t", "will not", sentence)

    sentence = re.sub(r"can\'t", "can not", sentence)

    sentence = re.sub(r"n\'t", " not", sentence)

    sentence = re.sub(r"\'re", " are", sentence)

    return sentence

def preprocessing(sentence):

    sentence = str(sentence)

    sentence = re.sub(r'xx\*', '', sentence)  # Removing XXXX

    sentence = re.sub(r'\d', '', sentence)  # Removing numbers

    sentence = sentence.strip(string.punctuation)

    tmp = ""

    for i in sentence.split(" "):  # Removing 2 letter words

        if i != 'no' or i != 'ct':

            tmp = tmp + ' ' + i

        prev = i

    # Replacing double space with single space

    tmp = re.sub(' {2,}', ' ', tmp)

    tmp = re.sub(r'\.+', ".", tmp)  # Replacing double . with single “.”

    tmp = tmp.lstrip()  # Removing space at the beginning

    tmp = tmp.rstrip()  # Removing space at the end

    return tmp

**7.3 Extracting Data from XML Report**

for filename in tqdm(os.listdir(dataset\_reports)):

    if filename.endswith(".xml"):

        f = os.path.join(dataset\_reports, filename)

        tree = ET.parse(f)

        root = tree.getroot()

        for child in root:

            if child.tag == 'uId':

                patient = child.attrib['id']

            if child.tag == 'MedlineCitation':

                for attr in child:

                    if attr.tag == 'Article':

                        for i in attr:

                            if i.tag == 'Abstract':

                                for name in i:

                                    if name.get('Label') == 'FINDINGS':

                                        finding = name.text

                                    if name.get('Label') == 'IMPRESSION':

                                        impression = name.text

                                    if name.get('Label') == 'INDICATION':

                                        indication = name.text

        for p\_image in root.findall('parentImage'):

            patient\_ids.append(patient)

            img.append(p\_image.get('id'))

            img\_finding.append(finding)

            img\_impression.append(impression)

            img\_indication.append(indication)

**7.4 Importing Dependencies**

import os

import cv2

import joblib

import numpy as np

import pandas as pd

import tensorflow as tf

from nltk.translate.bleu\_score import sentence\_bleu

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.layers import GlobalMaxPooling2D, Dropout, Add, MaxPooling2D, GRU, AveragePooling2D

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input, Embedding, LSTM, Dot, Reshape, Concatenate, BatchNormalization

tf.compat.v1.enable\_eager\_execution()

warnings.filterwarnings("ignore")

sns.set(palette='muted',style='white')

**7.5 Tokenizer Setup**

tokenizer = Tokenizer(filters = '',oov\_token = '<unk>')

tokenizer.fit\_on\_texts(train.impression\_final.values)

train\_captions = tokenizer.texts\_to\_sequences(train.impression\_final)

test\_captions = tokenizer.texts\_to\_sequences(test.impression\_final)

vocab\_size = len(tokenizer.word\_index)

caption\_len = np.array([len(i) for i in train\_captions])

start\_index = tokenizer.word\_index['<start>']

end\_index = tokenizer.word\_index['<end>']

**7.6 Input Data Pipeline**

class Dataset():

def \_\_init\_\_(self,df,input\_size,tokenizer = tokenizer, augmentation = True,  max\_pad = max\_pad):

self.img1 = df.image\_1

    self.img\_2 = df.image\_2

    self.caption = df.impression\_ip

    self.caption1 = df.impression\_op

    self.input\_size = input\_size

    self.tokenizer = tokenizer

    self.augmentation = augmentation

    self.max\_pad = max\_pad

    self.aug1 = iaa.Fliplr(1) # horizontal flip images

    self.aug2 = iaa.Flipud(1) # vertical flip images

  def \_\_len\_\_(self):

    return len(self.img1)

  def \_\_getitem\_\_(self,i):

    img1 = cv2.imread(self.img1[i],cv2.IMREAD\_UNCHANGED)/255

    img\_2 = cv2.imread(self.img\_2[i],cv2.IMREAD\_UNCHANGED)/255

    if img1.any()==None:

      print("%i , %s image sent null value"%(i,self.img1[i]))

    if img\_2.any()==None:

      print("%i , %s image sent null value"%(i,self.img\_2[i]))

    caption = self.tokenizer.texts\_to\_sequences(self.caption[i:i+1])

    caption = pad\_sequences(caption,maxlen = self.max\_pad,padding = 'post')

    caption = tf.squeeze(caption,axis=0)

    if self.augmentation:

          a = np.random.uniform()

          if a < 0.333:

              img1 = self.aug1.augment\_image(img1)

              img\_2 = self.aug1.augment\_image(img\_2)

          elif a < 0.667:

              img1 = self.aug2.augment\_image(img1)

              img\_2 = self.aug2.augment\_image(img\_2)

**7.7 ChexNet (Transfer Learning Model)**

def create\_chexnet(weights=chexnet\_weights, input\_size=(224, 224)):

    model = tf.keras.applications.DenseNet121(

        include\_top=False, input\_shape=input\_size+(3,))

    out = model.output

    out = GlobalAveragePooling2D()(out)

    out = Dense(14, activation="sigmoid", name="chexnet\_output")(out)

    chexnet = tf.keras.Model(inputs=model.input, outputs=out)

    chexnet.load\_weights(weights)

    chexnet = tf.keras.Model(

        inputs=model.input, outputs=chexnet.layers[-3].output)

    return chexnet

**7.8 Image Encoder Layer**

class Image\_encoder(tf.keras.layers.Layer):

    def \_\_init\_\_(self, name="image\_encoder"):

        super().\_\_init\_\_()

        self.chexnet = create\_chexnet(input\_size=(224, 224))

        self.chexnet.trainable = False

        self.avgpool = AveragePooling2D()

    def call(self, data):

        output = self.chexnet(data)

        output = self.avgpool(output)

        output = tf.reshape(

        output, shape=(-1, output.shape[1]\*output.shape[2], output.shape[3]))

        return output

**7.9 Image Encoder Block**

def encoder(img1, img2, dense\_dim, dropout\_rate):

    img\_enc = Image\_encoder()

    dense = Dense(dense\_dim, name='enc\_dense',

                  activation='relu')

    imf1 = img\_enc(img1)

    imf1 = dense(imf1)

    imf2 = img\_enc(img2)

    imf2 = dense(imf2)

    concat = Concatenate(axis=1)([imf1, imf2])

    bn = BatchNormalization(name="encoder\_batch\_norm")(concat)

    dropout = Dropout(dropout\_rate, name="encoder\_dropout")(bn)

    return dropout

**7.10 Global Attention Layer**

class global\_attention(tf.keras.layers.Layer):

    def \_\_init\_\_(self, dense\_dim):

        super().\_\_init\_\_()

        self.W1 = Dense(units=dense\_dim)

        self.W2 = Dense(units=dense\_dim)

        self.V = Dense(units=1)

    def call(self, encoder\_output, decoder\_h):

        decoder\_h = tf.expand\_dims(decoder\_h, axis=1)

        tanh\_input = self.W1(encoder\_output) + self.W2(decoder\_h)

        tanh\_output = tf.nn.tanh(tanh\_input)

        attention\_weights = tf.nn.softmax(self.V(tanh\_output), axis=1)

        output = attention\_weights\*encoder\_output

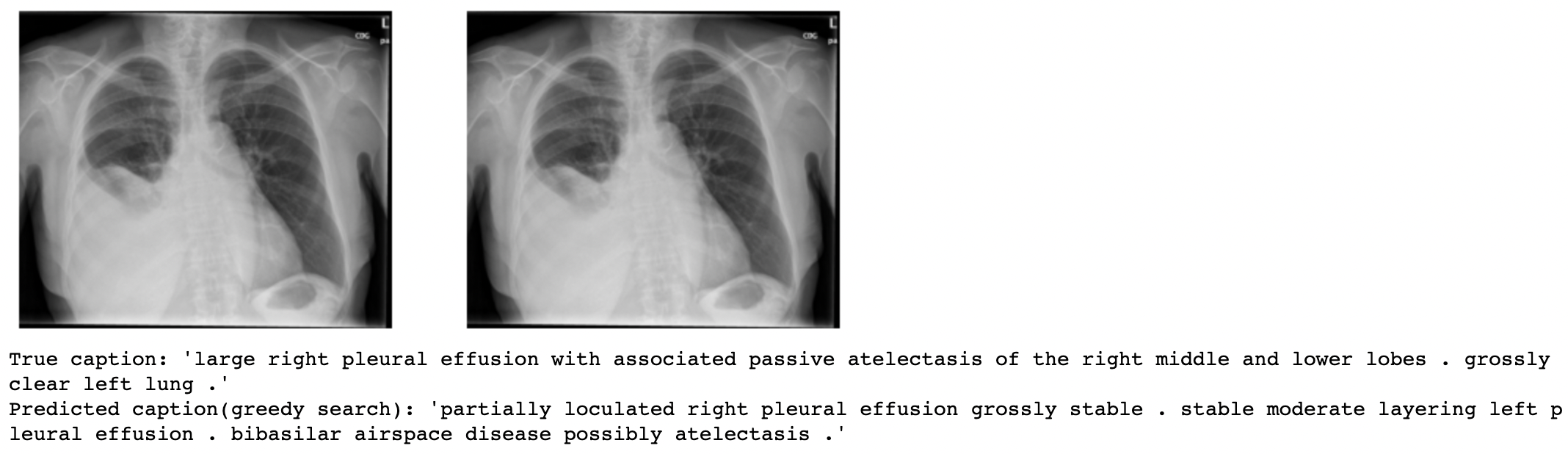
        context\_vector = tf.reduce\_sum(output, axis=1)

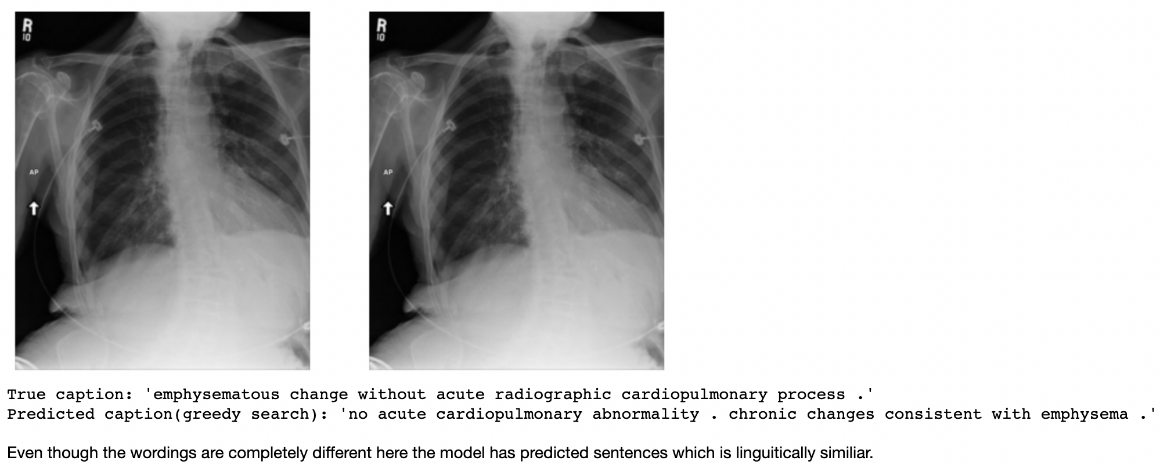
        return context\_vector, attention\_weights

# Results and Discussion

The automated generation of medical reports on chest X-ray images is highly viable, using a multi-step approach which employs the best techniques offered by deep learning and NLP. Chest X-rays are one of the most popular diagnostic tools and this project can improve the speed and quality of diagnosis. The CNN predicts the tags from the visual features and retains the spatial information in order to provide better context. The hierarchical LSTMs can decode the vector provided by the encoding layer in order to generate legible medical reports, which can be compared and evaluated using metrics such as BLEU Scores.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **BLEU1** | **BLEU2** | **BLEU3** | **BLEU4** |
| **Cascade RNN Model** [4] | 0.399 | 0.251 | 0.168 | 0.118 |
| **Co-attention model** [7] | 0.517 | 0.386 | 0.306 | 0.247 |
| **Simple Encoder** | 0.198 | 0.227 | 0.286 | 0.314 |
| **With Visual Attention** | 0.213 | 0.258 | 0.325 | 0.381 |





# References

[1] D. Demner-Fushman *et al.*, “Preparing a collection of radiology examinations for distribution and retrieval,” *J. Am. Med. Informatics Assoc.*, vol. 23, no. 2, pp. 304–310, 2016.

[2] P. Rajpurkar *et al.*, “CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning,” *arXiv*. 2017.

[3] A. Vaswani *et al.*, “Attention is all you need,” *Adv. Neural Inf. Process. Syst.*, vol. 30, pp. 5998–6008, 2017.

[4] H.-C. Shin, K. Roberts, L. Lu, D. Demner-Fushman, J. Yao, and R. M. Summers, “Learning to Read Chest X-Rays: Recurrent Neural Cascade Model for Automated Image Annotation,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2497–2506.

[5] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, “Image Captioning with Semantic Attention,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4651–4659.

[6] K. Xu *et al.*, “Show, attend and tell: neural image caption generation with visual attention,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, 2015, pp. 2048–2057.

[7] B. Jing, P. Xie, and E. Xing, “On the Automatic Generation of Medical Imaging Reports,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 2577–2586.

[8] T. Syeda-Mahmood *et al.*, “Chest X-Ray Report Generation Through Fine-Grained Label Learning,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2020, pp. 561–571.

[9] J. Yuan, H. Liao, R. Luo, and J. Luo, “Automatic radiology report generation based on multi-view image fusion and medical concept enrichment,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2019, pp. 721–729.

[10] Y. Xue *et al.*, “Multimodal recurrent model with attention for automated radiology report generation,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2018, pp. 457–466.

[11] G. Liu *et al.*, “Clinically accurate chest x-ray report generation,” in *Machine Learning for Healthcare Conference*, 2019, pp. 249–269.

[12] X. Huang, F. Yan, W. Xu, and M. Li, “Multi-attention and incorporating background information model for chest x-ray image report generation,” *IEEE Access*, vol. 7, pp. 154808–154817, 2019.

[13] M. Tanti, A. Gatt, and K. P. Camilleri, “Where to put the image in an image caption generator,” *Nat. Lang. Eng.*, vol. 24, no. 3, pp. 467–489, 2018.

[14] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global Vectors for Word Representation,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[15] Y. Zhang, Q. Chen, Z. Yang, H. Lin, and Z. Lu, “BioWordVec, improving biomedical word embeddings with subword information and MeSH,” *Sci. Data*, 2019.

[16] L. Gao, X. Li, J. Song, and H. T. Shen, “Hierarchical LSTMs with adaptive attention for visual captioning,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 5, pp. 1112–1131, 2019.

[17] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv Prepr. arXiv1409.0473*, 2014.

[18] Z. Yu, J. Yu, Y. Cui, D. Tao, and Q. Tian, “Deep Modular Co-Attention Networks for Visual Question Answering,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 6274–6283.